Bachelor's thesis

Business Information Technology

2024

Kirsi Ranki

Automated Deployment of Research Data Platform Exploits Reusability of Modular Components

– a professional article

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

Bachelor's Thesis | Abstract

Turku University of Applied Sciences

Business Information Technology

2024 | 19 pages

Kirsi Ranki

Automated Deployment of Research Data Platform Exploits Reusability of Modular Component

- a professional article

Research data platform is a data transfer pipeline, a transboundary service in central of the Wireless Communications and Cybersecurity research group's project organization. Development of the data platform have objectives related to the development of the platform itself as a product. However, there are other objectives too, including educational, exploratory, and innovational. Because of this, the platform is under continuous development process and in the center of the deployment process of the platform are service architecture and development environment, including installing, configuring, and launching the data pipeline service.

Deployment process with modular components provide multiple benefits for those working with platform. The process itself is easy, clear, static, and structured, every time the same, if needed. Modular components are flexible and reusable, with different cases of the deployment, whether there is a need for monolith structure or independent service to launch locally for testing purposes. Modularity provides benefits even further, right up to service productization, if desired.

This thesis is accomplished as a professional article, because of the partial recognition of studies and competence.

Keywords:
Deployment, modular components, research data platform

Opinnäytetyö (AMK) | tiivistelmä

Turun ammattikorkeakoulu

Tietojenkäsittely

2024 | 19 sivua

Kirsi Ranki

Tietoalustan automatisoitu käyttöönotto hyödyntää modulaarista komponenttirakennetta

- asiantuntija-artikkeli

Wireless Communications and Cybesecurity -tutkimusryhmä on kehittänyt osana tutkimushanketoimintaansa tietoalustan datan siirtoa varten. Hanketoiminnan vuoksi tietoalustalla on tuotekehityksen lisäksi koulutukseen, tutkimukseen ja innovointiin liittyviä tavoitteita, jotka näkyvät myös käyttöönoton prosessisen kehityksessä. Testausvaiheessa käyttöönoton prosessiin on sisällytetty palvelun arkkitehtuuri ja kehitysympäristö sisältäen asennukseen, konfigurointiin ja käynnistykseen liittyvät vaiheet.

Modulaarinen komponenttirakenne tarjoaa monenlaisia hyötyjä tietoalustan parissa työskenteleville. Prosessi itse on helppo, selkeä, staattinen ja strukturoitu, tarpeen vaatiessa joka käyttökerralla täysin samanlainen. Modulaariset komponentit ovat joustavia ja uudelleenkäytettäviä, mahdollistaen erilaisten yhdistelmien käyttöönoton käyttäjän tarpeiden mukaan. Modulaarisuus tarjoaa mahdollisuuksia myös jatkoon, aina käyttöönoton prosessin tuotteistamiseen (palvellullistaminen) asti.

Tämä työ on toteutettu osittaisen hyväksiluvun prosessin vuoksi asiantuntija-artikkelina.

Asiasanat:
Käyttöönotto, modulaarinen komponentti, tietoalusta

Content

Figures

1 Introduction

Deployment might sound simple, easy, and fast phase where system, service, software, server, functionality or any other set or part of the software infrastructure is prepared to be used. And simple it is, at least when the environment is static and stable in terms of straightforward development process, where aim is to proceed from plan to finished product.

However, projects that have other objectives, for example educational, exploratory, or innovational beside the development process, might face difficulties during deployment of unfinished, experimental, or floating parts of the infrastructure. To ease the usage of every part of the infrastructure in every phase of development process and lifecycle by anyone in interaction, there is an active testing process in progress where reusable and compatible components are used to construct modules for automated deployment process.

This thesis is accomplished as a professional article for the Wireless Communications and Cybersecurity research group because of the partial recognition of studies and competence.

2 Research data platform

Wireless Communication and Cybersecurity research group's one area of expertise is in maritime environments, with technologies and digitalization related to automated and autonomous maritime surface vessels and relative aspects and subdivisions [1]. Projects have covered a wide range of activities related to maritime testbeds and test environments for digital infrastructures [1] and one subdivision to work with has been developing research data platform, a data transfer and storing pipeline as transboundary service (Figure 1.) in central of the research group's project organization [2].

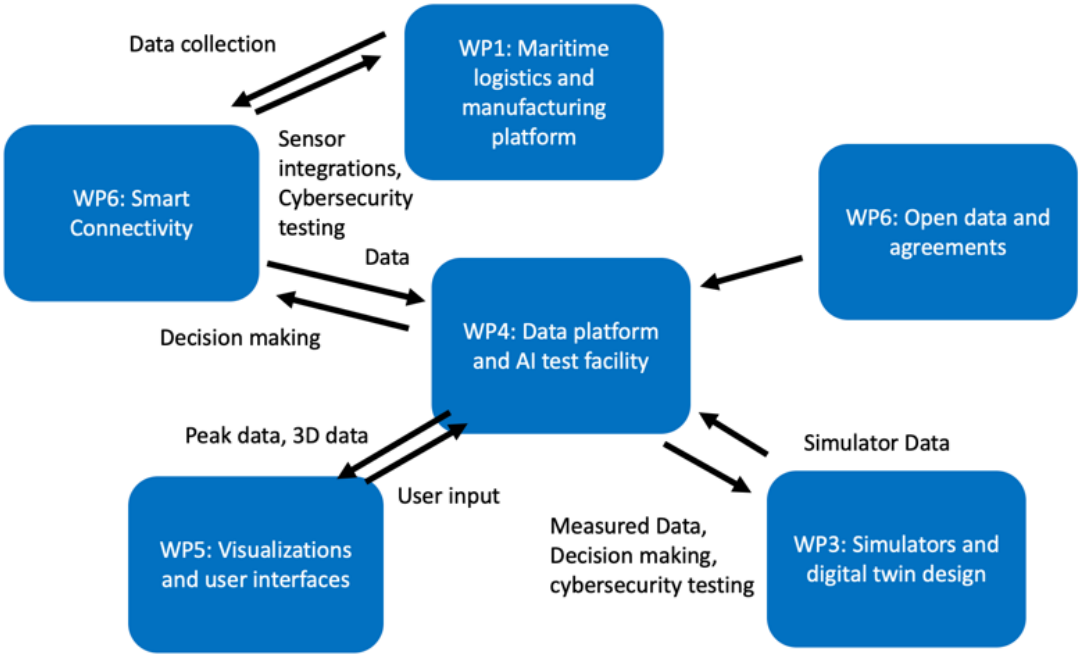

Figure 1. Work packages related to each other in ARPA project [2].

Origin of the research data platform is in the ARPA project, where core functions of the service were defined and developed in the years 2020 – 2023 [3]. After that, the development of the data platform has been implemented to research group's projects that exploit the features of the data platform. Data pipeline in platform includes receiving, handling, storing, and restoring the data.

Key features in the platform from the deployment point of view are the service architecture, development environment including core team with permanent workers of the project, subcontractors, and temporary workforce. Although deployment is quite often part of DevOps processes [4], and at least in theory, it usually includes planning and some kind of continuous CI/CD -pipeline [5, 6], in the end, it is a process where

servers, systems, services, applications, or codes go through pre-defined rules in certain order to install, configurate, and launch whatever deployable system component [7]. At this point automated deployment in the platform refers to installing, configuring, and launching the data pipeline service itself, not everyday development work based on coding.

3 Distributed service structure enables modularity in deployment

Architecture of the research data platform is constructed based on multiple services. These individually and independently working services are connected to each other via a message broker forming a seamless pipeline of data flow both in and out of the platform. Services forming the pipeline in the platform are hosted on several different virtual machines.

At this point, there are already seven (7) machines on the server side where each has a single service built and running. These services are responsible for pre-defined areas and functionalities of the data platform's features, each own one, and are a minimum viable architecture that supports the core actions of the platform in a way it is designed to (Figure 2).

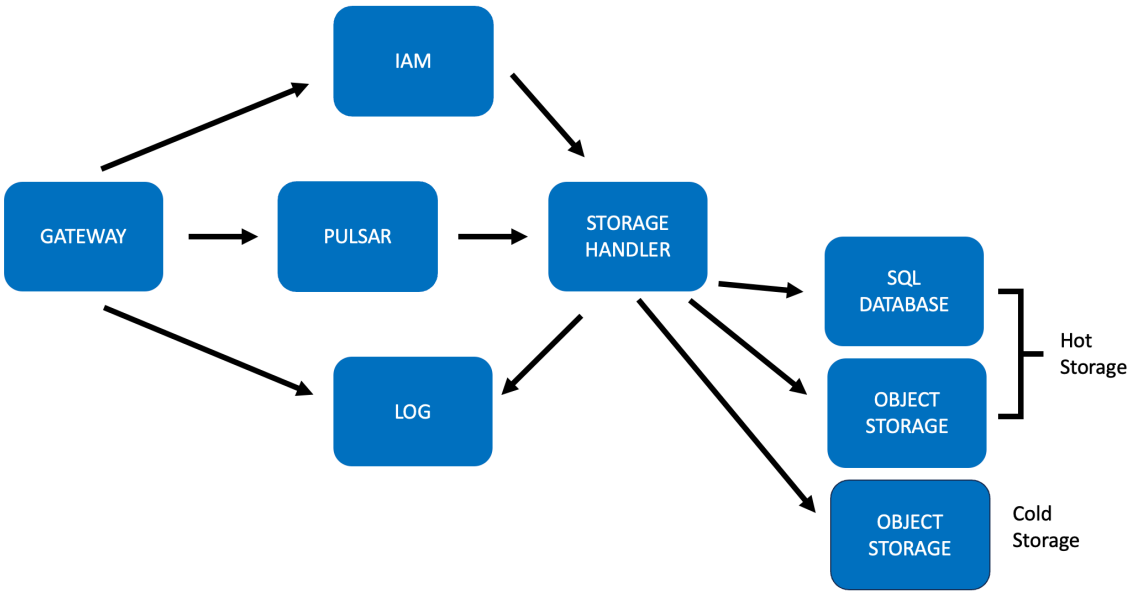

Figure 2. Architecture of the Research Data Platform.

Constructing each of these services requires setting up a proper virtual machine with necessary resources, installing needed applications and configuring supporting software's, dependencies, and service itself and of course, launching the service. In addition, some services require configuring, setting up, and launching custom utilities, scheduled tasks, and reoccurring jobs too.

Software projects come fast into situations, where manual deployment becomes too heavy and unnecessary challenging: the more there are elements to consider and dependencies affecting each other, the more demanding and time consuming the work

becomes. An appropriately functioning deployment process requires competence, knowledge, and understanding not only for service itself, but also for example a version compatibility and dependency management. Seamless workflow is not obvious, but rather an area of expertise to combine single working units to one functioning entity. [8, 9]

Research data platform has grown over the years of development process (Figure 3.) and preparing it for the usage requires a lot of resources that should be used efficiently. Continuing development and possible future scaling of the services will also multiply work that is needed to complete the deployment process every time it is done. Distribution of the services simplifies the deployment process in the platform by limiting the amount of interacting parts in the environment the service is functioning. Carefully completed preparations straightforward the processes and are prerequisite for stable ground deployment can be built on.

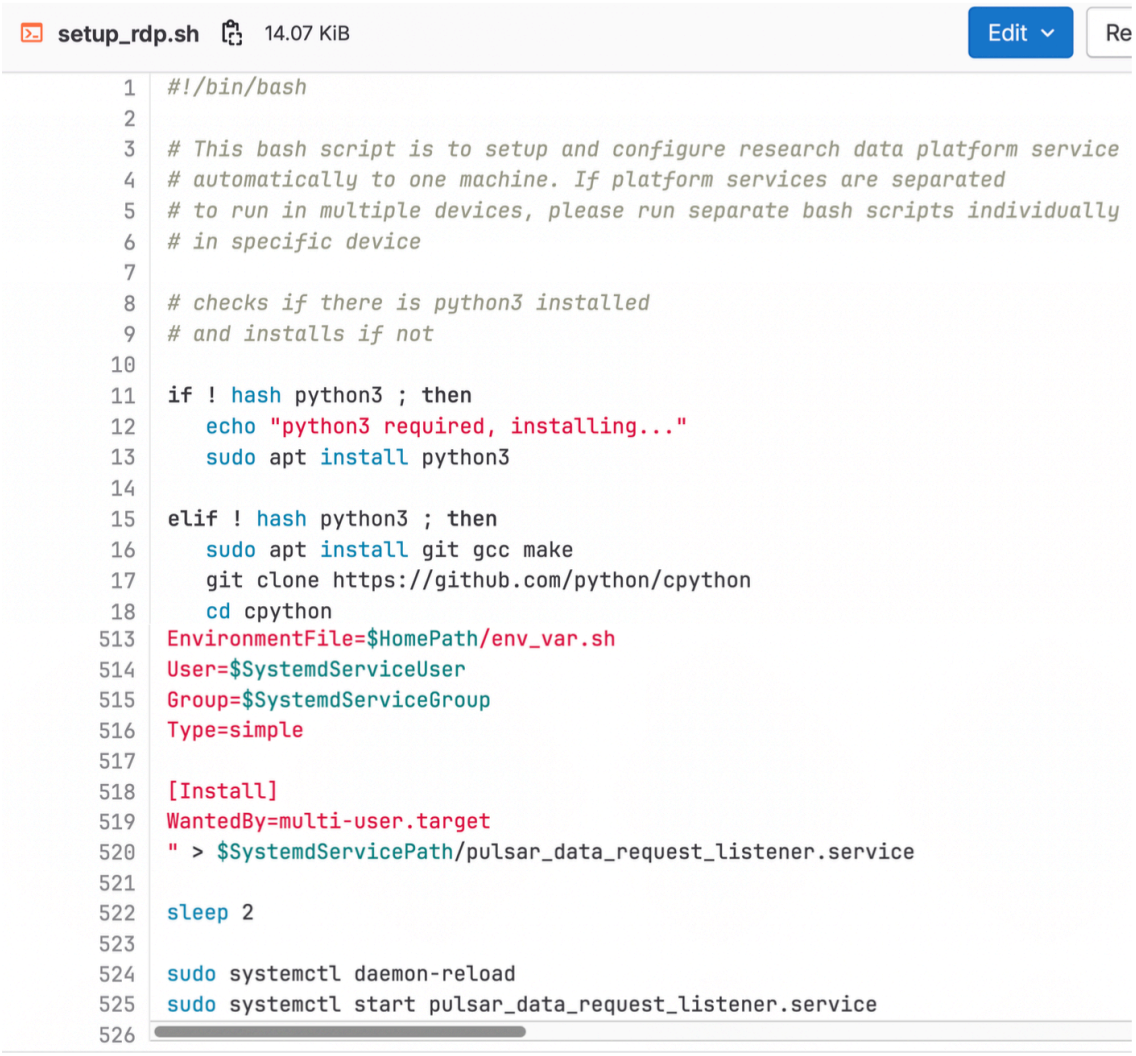

Figure 3. Example of deploying platform as monolith service, notice line amount of the script.

Service structure is quite complex with multiple different services to handle and there is always the possibility of further complexity with growing number of new services and servers when developing and scaling the platform for future needs. Automation enables the uniformity and predictability to the deployment that both exploits the carefully defined and prepared process and reduces expensive and unnecessary resource usage and overlapping and repetitive actions and work duties.

4 Deployment process follows structure constructed from modular components

Every phase of deployment has been constructed to its own module which can be used together to build components for the deployment process. Single module is the smallest part of the process and usually contains only one deployable activity (Figure 4). Single component is constructed based on modules needed to define, configure, set up, install, and launch one service of the platform to the proper environment. Most modules run their functionalities fully or at least mainly automatedly after user interference or programmatic triggers, but few of the modules require manual input from the user.

```
14  sudo apt update
15
16  INSTALL_APT_PACKAGES="python3-pip git net-tools curl"
17  for i in $INSTALL_APT_PACKAGES; do
18  |  sudo apt-get install -y $i
19  done
20
21  INSTALL_PYTHON_LIBRARIES="pulsar-client paho-mqtt config Flask"
22  for x in $INSTALL_PYTHON_LIBRARIES; do
23  |  yes | pip install $x
24  done
```

Figure 4. Example of dependencies -module from for storage handler service.

Deployment of the research data platform is enabled to be automated using two different tools: bash script (Figure 5) and Docker containers (Figure 6). Every service has its own deployment component with both tools but in addition, there is also a bash deployment component for installing the platform as a monolith service to one machine. All deployment alternatives use the same prerequisites and dependencies insofar as they are not service specific. Main difference between containers and bash scripts is the way services are executed and managed inside the deployable environment. Containers launch and maintain services itself, while bash scripts relay operating system's systemd service files when launching and managing services.

```
#!bin/bash

# Data source gateway deployment

sh ./python_installation.sh

source ./dsg_setup_conf.cfg

sh ./dsg_dependencies.sh

sh ./dsg_systemd.sh

```

Figure 5. Example of a bash deployment file from data source gateway service.

```
FROM ubuntu:22.04

ENV DEBIAN_FRONTEND=noninteractive

RUN mkdir /rdp
WORKDIR /rdp

RUN DEBIAN_FRONTEND=noninteractive && TZ=Europe/Helsinki && apt update && apt -y install tzdata
RUN apt update && apt install sudo

ADD . .

RUN bash dsg_dependencies.sh
RUN chmod 777 start_services.sh

ENTRYPOINT ["/bin/bash", "-c", "source RDPvenv/bin/activate && ./start_services.sh"]
```

Figure 6. Example of a Docker file from data source gateway service.

Requirements vary also between third party applications and customized (self-written) services. Configurations, utilities and other possible supportive applications and services are service specific too and require separate consideration in the deployment process. Despite of these service specific details, the main process in automated deployment follows a consistent format and therefore enables different phases of the process to be used simultaneously in different deployment cases.

5 Compile, prepare, deploy! Deployment in action

Although automated deployment is the main point of this article, in the case of the research data platform, the deployment process actually starts much earlier. When started from scratch, there needs to be machines to prepare and set up before any codes or programs can be executed. In addition to machines to host, the platform also needs an environment for where to run the deployment processes. These phases are not included in the automated deployment process of the platform itself, but process description contains literal instructions about the environment, computational resources and other prerequisites needed in order to run the services properly.

These first steps are done manually beforehand and after that, the deployment process of the service itself starts by preparing a proper environment. In Docker, this means launching the correct operating system first, and then preparing proper baseline installations that are called requirements in the data platforms deployment process. With bash installation, the environment is already prepared when setting up the machine(s) for the service and deployment starts with installing the needed requirements.

Requirements include installing correct programming language and runtime environment and for bash installations, also virtual environment to restrict and control service dependencies. At this step, also the possible and/or necessary users, groups, roles, and privileges are defined and created to be used by the service. When starting the deployment process, the user should first define and manually set the proper configurations for the service to predefined configuration files (Figure 7.).

```
28  # Location to the systemd service directory
29  SystemdServicePath="/lib/systemd/system"
30
31  # Configurations for service files owner privileges
32  SystemdChownUser=""
33  SystemdChownGroup=""
34
35  # Path to python3 and directory where script files are for service ExecStart and
    WorkingDir configuration
36  PythonPath="/usr/bin/python3"
37  ScriptsPath="~/research_platform/tsirp"
38
39  # Configurations for service user and group
40  SystemdServiceUser=""
41  SystemdServiceGroup=""
```

Figure 7. Example of a configuration file.

These simple text files contain local configuration parameters that vary based on for example use case and location of the service. Files have identical structure where every needed configuration is defined with key-value -pairs and these configurations are used

for example defining user credentials, necessary connection parameters (for example ip addresses and port numbers), and microservice identification for authorization and authentication purposes (for example credentials to create tokens).

Dependencies refer to both apt and pip packages required for the service to run properly. Administrators guarantee full and complete service functionality based on predefined conditions and therefore versions of the used packages are also defined carefully. Version upgrades are, of course, possible to be done by user, but it is important to notice that newest versions might have dependencies that are not included in version as it is currently in production.

Some services have custom utilities and other service-related tasks that need to be prepared and deployed too. These utilities have been written to own individual bash files that contain commands to prepare, install, construct, implement, and launch the task. Most of these tasks have a recurring nature and the script also contains commands to schedule utility tasks correctly. These files can be used both by container deployment and bash deployment.

Finally, the deployment process has a phase for service to be used as a part of the research data platform pipeline. In Dockerfiles, this executable command is set to ENTRYPOINT instructions for the service to be launched. In bash installation, all service components that construct the certain service, have been written to operational systems system software that executes the proper command(s) for launching the service.

6 Benefits of the modularity in deployment process

Deployment based on modular components sounds great when it is written down here like this with all the good assumptions brought up. But does it actually work as well in practice as in paper? Being now in test use at the platform, there are of course questions related to its functioning that can not yet be answered. But based on usage this far, there are already multiple factors that advocate continuing to test and develop work still further.

Of course, one main advantage is easy, clear, static, and structured deployment. When necessary, the deployment process is always exactly the same and duplicable as it is. This is crucial especially in a production environment, when service is dependent on a certain environment, applications, third party services, installations, configurations, and specific versions.

The dysfunctional parts of the services and single service can be built repeatedly time after time and just like they were originally, without the need to spend resources to construct the instructions for the deployment process again every time. This is of course, a benefit of the deployment process even without modularity. The extra layer modular components provide to the deployment is a much easier and better management of sub-services and -processes whenever there is no need to start everything from a clean slate. Deployment process consists of the latest information about apt and pip packages with appropriate version information included.

Files related to deployment and installation process are part of software repository which means that it is included to the software's version control automatically (Figure 8.). This ensures the process is always combined to the latest version of the service in production. Administrators and developers have, of course, the responsibility to up to date both the service and the process simultaneously, but here documentation have an important role, but also the habit to do so and understanding about the critical part deployment and installation has in services' overall architecture.

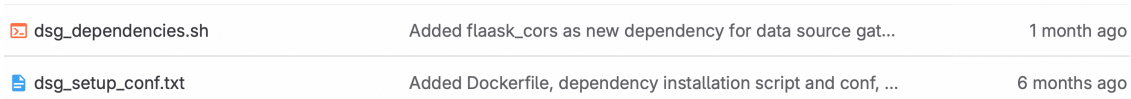

Figure 8. Example of deployable modules in GitLab version control.

Development work of the data platform is divided into a few different sub-divisions in work tasks and turnover of the developers is quite high, at least outside the core team of expertise. For example, there are hundreds if not thousands of hours of development work done for the platform annually by students. These mutually beneficial co-operations are usually limited in terms of time and that's why assignments are usually quite strict, concerning specific parts of the service and/or specific functionality. Independent service structure makes it possible to use only those services that are

crucial when developing that certain assignment. Especially with temporary or part-time workforce it is important that everyone can get started with the work itself without the need to worry about how to get everything running at first.

When deployment is fast and easy, precious resources are not wasted on understanding the whole architecture, structure, and concept of the research data platform at once, but one part at a time. Modularity in deployment also reduces need for resources, when development work is done locally, because there is no need for installing and deploying the whole platform to a local machine, only those parts and services that are needed to complete certain assignments. With this, it is possible to significantly reduce the amount of information at a time, focus on essentialities of certain tasks and increase the knowledge in a controlled manner. Familiarization to the parts included in deployment will happen during work to the extent that it is necessary in everyone's individual situation.

7 Continuous development of the deployment process

Like said above, the automated deployment process based on modular components is under constant development, just like the service(s) to which it applies. However, that is not all development work related to it. The process itself is under continuous development and there are objectives to improve it still further.

The further platform as a service is proceeding and the more it grows within, the more there are requirements related to the deployment. Process itself needs resources, both computational and human and the wider it gets, the more resources it will use. Updating and maintaining the process takes its own share of the work. There are expectations for the deployment process to give more to the project than it takes, and one way to increase the efficiency is to modify the use cases of the automated deployment further.

At least in theory, modularity of the deployment allows usage in as many ways as there are single components to use either separately or in various combinations. Of course, every customization demands manual intervention from user at least with configurations, but technically different parts of the platform can be put into service anywhere. For example, parts of the service can be installed and launched to be used in local machines of research groups test vessel eM/S Salama [10, 11], that is used to collect maritime data. With that, some of the pipeline processes could be used instantly at the time data is collected in the environment collection is done because there is not yet a possibility to transfer the data over the network.

One possible direction for the development work and goal for the process could be service productization. With bash deployment, it is already possible to write modules to accept user inputs and, in that way, to be more self-directive and user friendly. This requires more work from developers to the background of the process, but it might fasten and simplify the use and the process itself significantly. There are a lot of third-party tools to be used for the deployment process already, so the direction of the deployment might be reasoned in the future.

References

[1] Turku AMK 2024. Wireless Communications and Cybersecurity. https://www.turkuamk.fi/tutkimusryhmat/wireless-communications-and-cybersecurity/.

[2] Tuomola, T., Kalliovaara, J. 2022. Data platform enables open access to the data collected in ARPA. ARPA blog 20.6.2022. https://arpa-project.turkuamk.fi/yleinen/data-platform-enables-open-access-to-the-data-collected-in-arpa/.

[3] Paavola, J., Kivelä, S. 2023. Development of Applied Research Platforms for Autonomous and Remotely Operated Systems. Reports from Turku University of Applied Sciences 294. https://urn.fi/URN:ISBN:978-952-216-862-7.

[4] Diaz, H. 2024. Automate Deployment in DevOps with DevOps tools: The Ultimate Guide. Dev blog 8.8.2024. https://dev.to/harman_diaz/automate-deployment-in-devops-with-devops-tools-the-ultimate-guide-1a1b.

[5] Thakker, H., 2023. Software deployment Best Practices in 2023. Medium article 6.8.2023. https://medium.com/@hardikthakker/software-deployment-best-practices-4a0c0e0af65b.

[6] Belagatti, P. 2023. From Code to Deployment: A Complete Hands-On Guide to CI/CD! LinkedIn article 3.4.2023. https://www.linkedin.com/pulse/from-code-deployment-complete-hands-on-guide-cicd-pavan-belagatti.

[7] Alkhabbas, F. et al., 2019. Characterizing Internet of Things Systems through Taxonomies: A Systematic Mapping Study. Internet of Things. https://www.sciencedirect.com/topics/computer-science/deployment-process.

[8] Bala, G. 2024. DevOps – Part 4 – Finally, deploying! LinkedIn article 27.6.2024. https://www.linkedin.com/pulse/devops-part-4-finally-deploying-greg-bala-cvqac.

[9] Poddar, R. 2017. Automated Software Deployment – Benefits Simplified. LinkedIn article 14.5.2017. https://www.linkedin.com/pulse/automated-software-deployment-benefits-simplified-ranjit-poddar?trk=public_profile_article_view.

[10] Kalliovaara, J. et al., 2024. Deep Learning for Maritime Applications: Development of the eM/S Salama Unmanned Surface Vessel and its Remote Operations Center as a Test Platform for Sensor Data Collection and Algorithm Development. Remote Sens. 2023, 1, 0. https://www.researchgate.net/publication/379270272_Deep_Learning_for_Maritime_Applications_Development_of_the_eMS_Salama_Unmanned_Surface_Vessel_and_its_Remote_Operations_Center_as_a_Test_Platform_for_Sensor_Data_Collection_and_Algorithm_Development.

[11] Turku AMK 2024. Autonomous and Intelligent Systems (AIS) Laboratory. https://www.turkuamk.fi/en/service/autonomous-and-intelligent-systems-laboratory